DYNAMIC OBJECT VISUALIZATION AND CODE GENERATION

BACKGROUND

The present invention relates to business intelligence tools for building applications on a database management system (DBMS).

The advent of powerful, yet economical computers made possible by advances in processor, memory and data storage devices has made computers an integral part of modern companies. An important class of application for these computers includes a DBMS where information is collected and organized according to a data model and searched using queries. The DBMS allows users to perform operations such as locating, adding, deleting and updating records stored in the computer without a detailed knowledge of how the information making up the records actually is stored in the computer.

One powerful type of DBMS is known as a relational DBMS where stored information appears to the user as a set of tables, each of which is termed a relation. In each relation, the information appears to be arranged in rows and columns, with columns of data being related to each other by one or more predetermined functions.

To access particular information in the relational DBMS, a query compiler converts a user request, typically expressed in a query language such as a Structured Query Language (SQL), into a set of operations to be performed on one or more input relations to yield a solution responsive to the user's request. Using the query language provided by the DBMS, the user may develop application programs which facilitate retrieval of the data from the DBMS, processing of the data, and organization of the data into reports.

One issue in developing business intelligence tools is the type of reports that the tool is to generate. Typically, the tool generates certain pre-formatted reports using the query language. Although the query language is easier to use then conventional programing languages such as Basic or C, the generation of each new report still requires a certain programming expertise and can often take a substantial amount of time.

SUMMARY

The invention provides user access to data through information spaces called scenes that allow the user to understand, view and navigate data. In one aspect, a method for executing an application expressed as a scene graph is disclosed. The scene graph is a hierarchical representation of one or more objects, each object capable of generating code associated with the object. The application is executed by traversing the hierarchy of the scene graph; and for each object stored in the scene graph, instructing each object to self-execute.

Implementations of the invention include the following. The method generates an execution image for each scene of the scene graph by: retrieving byte code associated with each node of the object; storing the byte code as part of the scene; and generating the byte code execution image for the scene. Further, the node is characterized as one of several types. If the node type is a shape type, the method generates a shape creation statement; generates a begin property statement; for each property, generates a statement to set each property associated with the object; and generates a statement to commit each property. If the node type is a data source type, the method generates a database query; determines column names for referenceable identifiers; generates a statement to create the query; for each parameter, creates a statement to set a query parameter value; and generates a statement to execute the query. The byte code image is then provided to an execution engine. An execution context is then created for the byte code.

When the statement in the byte code is executed, the method determines whether the statement is a create shape statement, and if so: creating the shape; and storing the shape in a context hash table using a tokenized shape name. If the statement is a begin property statement, the method searches for the shape in a context hash table; and executes a begin method associated with the shape. If the statement is a set property statement, the method searches for the shape in a context hash table; evaluates a property expression associated with the shape; and assigns a value to the property. If the statement is an end property statement, the method searches for the shape in a context hash table; executes an end properties method associated with the shape to commit the properties; initializes the shape; and adds the shape to a display list. The method also refreshes a canvas associated with the shape. Each property of the object is set, and the object is notified before and after the setting the property of the object.

In another aspect, an editor for visually editing data representation is disclosed. The data has a parent graph and one or more graphical data elements recursively nested in the parent graph. The editor has a window for editing a parent graph and one or more drill-down windows for editing each of the one or more graphical data elements nested in the parent graph, each of the graphical data elements containing a nested graph.

Implementations of the editor includes one or more of the following. An attribute window is associated with each of the graphical data elements. The attribute window is used to edit properties associated with a computer-implemented object having an object state and one or more interfaces providing access to the object state through a plurality of attributes, each of the attributes defined as a functional expression and referenceable at run-time as a data value. The functional expression includes one or more of the following: a function; an operator; a database column name; a variable; and a constant. The attribute may be a static data value. The functional expression may be parsed to generate a function which is stored as a run-time value. The function may be cloned and stored as a design time value if the function is a constant. Further, an error message may be displayed if the expression is invalid.

Advantages of the invention include one or more of the following. The invention is a visual business intelligence tool for building applications that extend beyond the limitations inherent in conventional forms-based or report-based applications. Specialized programmers are removed from the application development process and users are moved closer to the data so that application development time is reduced. User interfaces can be created quickly and easily for information rich databases and for applications such as data warehousing and decision support.

The invention's hyperlinks provides context and "look-ahead" information to applications. This capability supports several powerful advantages in building data-driven applications. First, users can see through portals into other views of their data without losing the context of where they are. The navigational path taken by a user browsing the application can affect the application itself, thus providing dynamic customization of the application. In addition, con-